A Generic Approach Towards Image Manipulation Parameter Estimation Using Convolutional Neural Networks

Belhassen Bayar
Drexel University
Dept. of Electrical & Computer Engineering
Philadelphia, PA, USA
bb632@drexel.edu

Matthew C. Stamm
Drexel University
Dept. of Electrical & Computer Engineering
Philadelphia, PA, USA
mstamm@coe.drexel.edu

ABSTRACT

Estimating manipulation parameter values is an important problem in image forensics. While several algorithms have been proposed to accomplish this, their application is exclusively limited to one type of image manipulation. These existing techniques are often designed using classical approaches from estimation theory by constructing parametric models of image data. This is problematic since this process of developing a theoretical model then deriving a parameter estimator must be repeated each time a new image manipulation is derived. In this paper, we propose a new data-driven generic approach to performing manipulation parameter estimation. Our proposed approach can be adapted to operate on several different manipulations without requiring a forensic investigator to make substantial changes to the proposed method. To accomplish this, we reformulate estimation as a classification problem by partitioning the parameter space into disjoint subsets such that each parameter subset is assigned a distinct class. Subsequently, we design a constrained CNN-based classifier that is able to extract classification features directly from data as well as estimating the manipulation parameter value in a subject image. Through a set of experiments, we demonstrated the effectiveness of our approach using four different types of manipulations.

KEYWORDS

Image forensics; manipulation parameter estimation; convolutional neural networks; quantization

1 INTRODUCTION

Digital images play an important role in a wide variety of settings. They are used in news reporting, as evidence in criminal investigations and legal proceedings, and as signal intelligence in governmental and military scenarios. Unfortunately, widely available photo editing software makes it possible for information attackers to create image forgeries capable of fooling the human eye. In order to regain trust in digital images, researchers have developed a wide variety of techniques to detect image editing and trace an image's processing history [36].

IH&MMSec '17, June 20–22, 2017, Philadelphia, PA, USA
© 2017 ACM. ISBN 978-1-4503-5061-7/17/06…$15.00.
DOI: http://dx.doi.org/10.1145/3082031.3083249

An important part of characterizing an image's processing history involves determining specifically how each editing operation was applied. Since several editing operations used to manipulate an image are parameterized, this involves estimating these manipulation parameters. For example, a user must choose a scaling factor when resizing an image, a quality factor when compressing an image, or blur kernel parameters (e.g. kernel size, blur variance) when smoothing an image.

Estimating manipulation parameter values may also be important when performing several other forensics and security related tasks. In some cases, it is useful or necessary to determine manipulation parameter values when detecting the use of multiple editing operations and tracing processing chains [10, 32]. Manipulation parameter estimates can be used to undo the effects of editing or provide an investigator with information about an image before it was edited. They can also be used to improve camera identification algorithms [16] or used as camera model identification features [21]. Additionally, manipulation parameter estimates can be used to increase the performance of some steganographic algorithms [26] and watermark detectors [11, 30].

Existing manipulation parameter estimation algorithms are often designed using classical approaches from estimation theory. This is typically done by first constructing a theoretical model to describe a manipulated image or some image statistic (e.g. pixel value or DCT coefficient histograms) that is parameterized by the manipulation parameter that is to be estimated. Next, an estimator for the manipulation parameter is theoretically derived from the statistical model. Algorithms have been developed to estimate the scaling factor used when resizing an image [27, 28], the contrast enhancement mapping applied to an image [13, 33, 34], the quality factor or quantization matrix used when compressing an image [6, 12, 26, 37], the size of the filter window used when median filtering an image [20], and blurring kernel parameters [1, 7, 9].

While classical approaches from estimation theory have led to the development of several successful manipulation parameter estimation algorithms, developing estimation algorithms for new manipulations or improving upon existing algorithms can be quite challenging. It is frequently difficult to develop accurate parametric models of image data that can be used for manipulation parameter estimation. Once a model is constructed, deriving a manipulation parameter estimator from this model may also be both difficult and time consuming. Furthermore, this process of developing a theoretical model then deriving a parameter estimator must be repeated each time a new image manipulation is developed.

In light of these challenges, it is clear that forensic researchers can benefit from the development of a *generic* approach to performing manipulation parameter estimation. By generic, we mean an approach that can be easily adapted to perform parameter estimation for different manipulations without requiring an investigator to make anything other than minor changes to the estimation algorithm. Instead of relying on theoretical analysis of parametric models, this approach should be data-driven. In other words, this approach should be able to *learn* estimators directly from a set of labeled data.

Recent work in multimedia forensics suggests that this goal may be accomplished by using convolutional neural networks (CNNs). CNNs have already been developed to learn image manipulation detectors directly from data. For example, we showed in our previous work that by incorporating a "constrained convolutional layer" into the beginning of a CNN architecture, we could train this fixed architecture to detect several different image manipulations [2, 3]. Similarly, Chen et al. showed that a CNN can be trained to perform median filtering detection using an image's median filter residual [8].

In this paper, we propose a new, generic data-driven approach to performing manipulation parameter estimation. Our approach does not require researchers to develop a parametric model of a particular manipulation trace in order to construct an estimator. Instead, manipulation parameter estimation features are learned directly from training data using a constrained convolutional neural network. Furthermore, our CNN can be re-trained to perform parameter estimation for different manipulations without requiring changes to the CNN's architecture except for the output classes. Our approach operates by first approximately reformulating manipulation parameter estimation as a classification problem. This is done by dividing the manipulation parameter set into different subsets, then assigning a class to each subset. After this, our specially designed CNN is used to learn traces left by a desired manipulation that has been applied using parameter values in each parameter subset. We experimentally evaluated our proposed CNN-based estimation approach using four different parameterized manipulations. The results of our experiments show that our proposed approach can correctly identify the manipulation parameter subset and provide an approximate parameter estimate for each of these four manipulations with estimation accuracies typically in the 95% to 99% range.

The remainder of this paper is organized as follows. In Section 2, we provide an overview of our proposed generic parameter estimation approach, including details on class formation and high-level classifier design. Our parameter estimation CNN architecture is described in detail in Section 3. In Section 4, we experimentally evaluate the performance of our proposed approach when performing parameter estimation for four different manipulations: resizing, JPEG compression, median filtering, and Gaussian blurring. Section 5 concludes this paper.

2 PROPOSED ESTIMATION APPROACH

To develop our CNN-based approach to performing manipulation parameter estimation, we begin by assuming that an image under investigation I is a manipulated version of some original image I' such that

$$I = m(I', \theta) \tag{1}$$

where $m(\cdot)$ is a known image editing operation that is parameterized by parameter θ. We assume that the set of possible parameter values Θ is totally ordered and is known to the forensic investigator. In this paper, we assume that θ is one dimensional (e.g. a scaling factor or JPEG compression quality factor), however it is simple to extend our approach to the case of multidimensional θ's.

While the editing operation m is known to an investigator, we do not assume that the investigator knows the specific traces left by m. We do assume, however, that m leaves behind traces that are learnable by some classifier g. Prior research has shown that traces left by several manipulations can be learned using CNNs construced using a constrained convolutional layer [3] or by an ensemble classifier provided with rich model features [14, 29]. Furthermore, we assume that the specific nature of these traces changes depending on the choice of the manipulation parameter θ. Additionally, we assume that an investigator has access to a large corpus of images and can modify them with m using different values of θ in order to create training data for our parameter estimator.

2.1 Formulating parameter estimation as a classification problem

In order to leverage the power of CNNs that are able to learn manipulation traces, we first approximately reformulate our parameter estimation problem as a classification problem. To do this, we partition the parameter set Θ into K disjoint subsets ϕ_k such that

$$\phi_k = \{\theta : t_k \leq \theta < t_{k+1}\}, \tag{2}$$

for $k = 1, \ldots, K-1$ and

$$\phi_K = \{\theta : t_K \leq \theta \leq t_{K+1}\}, \tag{3}$$

where t_1 is the smallest element in Θ, t_{K+1} is the largest element in Θ, and the t_k's form upper and lower boundaries for each of the parameter subsets. Taken together, the set Φ of all subsets ϕ_k form a minimal cover for Θ, i.e.,

$$\bigcup_{k=1}^{K} \phi_k = \Theta. \tag{4}$$

When constructing our CNN-based classifier g, each parameter subset is assigned a distinct class label c_k such that

$$g(I) = c_k \quad \Rightarrow \quad \theta \in \phi_k. \tag{5}$$

Figure 1 shows an overview of how parameter subsets and their corresponding classes are formed by partitioning the parameter space. If we wish to include the possibility that the image is unaltered (i.e. no parameter value can be estimated because m hasn't been used to modify I), then an additional class c_0 can be added to the classifier to represent this possiblity.

To produce parameter estimates, we construct an additional function $h(\cdot)$ that maps each class to an estimated parameter value $\hat{\theta}$. The function h can be constructed in multiple ways depending on the nature of the paramter subsets ϕ_k.

For some estimation problems where Θ is finite and countable, each parameter subset can be chosen to contain only one element. Examples of this include estimating the window size of a median filter or the quality factor used when performing JPEG compression.

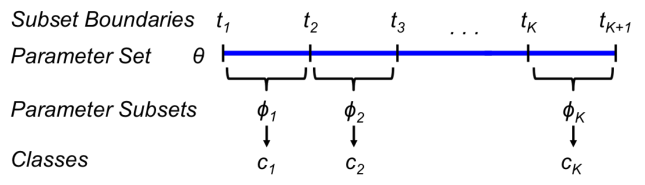

Figure 1: Overview of how classes are formed by partitioning the parameter set.

Since each parameter subset contains only one element, the parameter estimate is chosen to be the lone element of the parameter subset that corresponds to the class chosen by the classifier, i.e.

$$\hat{\theta} = \theta_k \quad \text{given } \phi_k = \{\theta_k\}. \tag{6}$$

In other estimation scenarios, each parameter subset may be chosen to contain multiple elements or may be uncountable. Estimating the scaling factor used when resizing an image is a typical example of this, since the set of possible scaling factors itself is uncountable. In these cases, a parameter estimate produced by the equation

$$\hat{\theta} = \frac{t_k + t_{k+1}}{2}. \tag{7}$$

This is equivalent to choosing the parameter estimate as the centroid of the parameter subset that corresponds to the class chosen by the classifier.

Both rules (6) and (7) can be taken together to produce the parameter estimation function

$$\hat{\theta} = h(c_k) = \begin{cases} \theta_k & \text{if } \phi_k = \{\theta_k\}, \\ \frac{t_k + t_{k+1}}{2} & \text{if } |\phi_k| \neq 1. \end{cases} \tag{8}$$

We note that our approach can be roughly interpreted as choosing between several quantized values of the manipulation parameter θ. While quantization will naturally introduce some error into the final estimate produced, it allows us to define a finite number of classes for our classifier to choose between. The estimation error introduced by this quantization can be controlled by decreasing the distance between the class boudaries t_k at the expense of increasing the number of classes.

2.2 Classifier design

For the estimation approach outlined in Section 2.1 to be truly generic, we need to construct a classifier $g(\cdot)$ that is able to directly learn from data some parameter specific traces left by a manipulation m. This requires the use of some generic low-level feature extractors that can expose traces of many different manipulations. While CNNs are able to learn feature extractors from training data, CNNs in their standard form tend to learn features that represent an image's content. As a result, they must be modified in order to become suitable for forensic applications. To create our CNN for performing manipulation parameter estimation, we leverage significant prior research that shows that traces left by many different manipulations can be learned from sets of prediction residuals [8, 19, 22, 24, 29].

Prediction residual features are formed by using some function $f(\cdot)$ to predict the value of a pixel based on that pixel's neighbors within a local window. The true pixel value is then subtracted from the predicted value to obtain the prediction residual r such that

$$r = f(I) - I \tag{9}$$

Frequently, a diverse set of L different prediction functions are used to obtain many different residual features. Many existing generic feature sets used in forensics take this form, including rich model features [14, 29], SPAM features [24, 25], and median filter residual features [8, 19]. These prediction residual features suppress an image's contents but still allow traces in the form of content-independent pixel value relationships to be learned by a classifier.

To provide our CNN-based classifier g with low-level prediction residual features, we make our CNN's first layer a *constrained convolutional layer* [3]. This layer is formed by using L different convolutional filters w_ℓ that are adaptively learned, but are constrained to be prediction error filters. These filters are initially seeded with random values, then their filter weights are iteratively learned through a stochastic gradient descent update during the backpropagation step of training. Since this update may move each filter outside of the set of prediction error filters, the following constraints

$$\begin{cases} w_\ell(0,0) = -1, \\ \sum_{m,n \neq 0} w_\ell(m,n) = 1, \end{cases} \tag{10}$$

are enforced upon each filter immediately after the backpropagation step to project the updated filter back into the set of prediction error filters.

It can easily be shown that the L feature maps produced by a constrained convolutional layer are residuals of the form (9). A simple way to see this is to define a new filter $\tilde{w}_\ell$ as

$$\tilde{w}_\ell(m,n) = \begin{cases} w(m,n) & \text{if } (m,n) \neq (0,0), \\ 0 & \text{if } (m,n) = (0,0). \end{cases} \tag{11}$$

As a result, the feature map produced by convolving an image with the filter w_ℓ is

$$r_\ell = w * I = \tilde{w}_\ell * I - I. \tag{12}$$

By defining $f(I) = \tilde{w}_\ell * I$, we can see these residuals are of the same form as in (9).

By using a constrained convolutional layer, our CNN can be trained to learn appropriate residual features for estimating parameter values associated with different manipulations instead of relying on fixed residual features. Associations between these features are learned by higher layers of our CNN, whose full architecture is described below in Section 3.

The final layer of our CNN consists of K neurons with a softmax activation function. Each neuron corresponds to a unique class (and its associated parameter set) defined in Section 2.1. A class c is chosen by our CNN according to the rule

$$c = \arg\max_k \lambda_k, \tag{13}$$

where λ_k is the activation level of the neuron corresponding to the k^{th} class. Since we use a softmax activation function, the activation levels in the last layer can be loosely interpreted as a probability distribution over the set of classes. As a result, (13) can be loosely interpreted as choosing the most probable class.

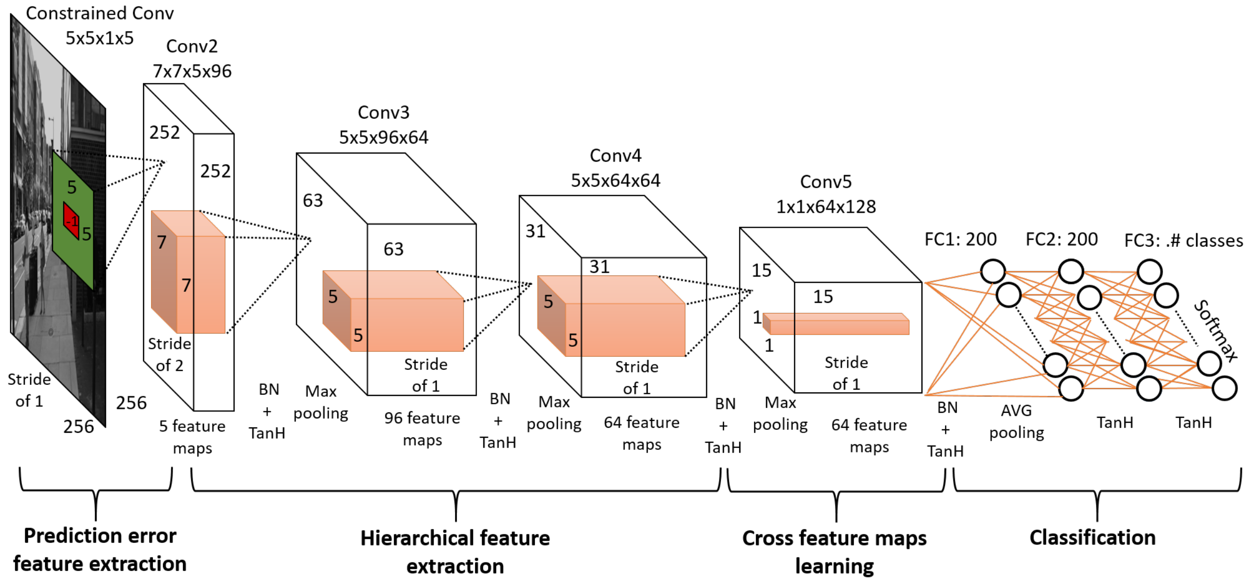

Figure 2: CNN proposed architecture; BN:Batch-Normalization Layer; TanH: Hyperbolic Tangent Layer

Parameter estimation steps. Our proposed method is summarized below.

```
Input: Image $I$ manipulated by $m(\cdot)$ and parameter set $\Theta$.
Output: Estimated parameter $\hat{\theta}$.

  Step 1  Partition the parameter space $\Theta$ into a set of disjoint
          subsets $\phi_k$ defined in Eqs. (2) and (3) to form a cover for $\Theta$.
  Step 2  Define a CNN-based classifier $g$ in Eq. (5) such that
          each parameter subset is assigned a distinct class $c_k$.
  Step 3  Define the estimate $\hat{\theta}$ as denoted in Eq. (8).
  Step 4  Train a constrained CNN to classify input images into
          the set of classes $c_k$'s.
  Step 5  Estimate $c = \arg\max_k \lambda_k$ where $\lambda_k$ is the activation
          level of the neuron corresponding to the kth class in CNN.
  Step 6  Assign $\hat{\theta} = h(c)$.
```

3 NETWORK ARCHITECTURE

In this section, we give an overview of the constrained CNN architecture used in our generic parameter estimation approach. We note that our CNN architecture differs significantly from the architecture proposed in [3]. Fig. 2 depicts the overall architecture of our CNN. One can observe that we use four conceptual blocks to build a CNN's architecture capable of distinguishing between different manipulation parameters.

Our proposed CNN has different conceptual blocks designed to: (1) jointly suppress an image's content and learn low-level pixel-value dependency features while training the network, (2) learn higher-level classification features through deeper convolutional layers and (3) learn associations across feature maps using 1×1 convolutional filters. These 1×1 filters are used to learn linear combination between features located at the same spatial location but belong to different feature maps [4]. The input to the CNN is a grayscale image (or a green color layer of an image) patch sized 256×256 pixels. In what follows, we give more details about each block.

3.1 Pixel-value dependency feature extraction

As mentioned in Section 2.2, CNNs in their existing form tend to learn features related to an image's content. If CNNs of this form are used to identify parameters of an image manipulation, this will lead to a classifier that identifies scene content associated with the training data. To address this problem, in our architecture we make use of a constrained convolutional layer [3] ("Constrained Conv") which is able to jointly suppress an image's content and learn pixel-value dependency traces induced by one particular manipulation's parameter. This layer consists of five constrained convolutional prediction-error filters of size 5×5 adaptively learned while training the CNN and operates with a stride of size 1. The output of this "Constrained Conv" layer will take the form of prediction-error feature maps of size 252×252×5. These residual features are vulnerable to be destroyed by nonlinear operations such as, activation function and pooling layer. Therefore, they are directly passed to a regular convolutional layer.

3.2 Hierarchical feature extraction

In our second conceptual block, we use a set of three regular convolutional layers to learn new associations and higher-level prediction-error features. From Fig. 2, one can notice that all convolutional layers in the network operate with a stride of 1 except "Conv2 layer which uses a stride of 2. We also can notice that all the three convolutional layers are followed by a batch normalization (BN) layer. Specifically, this type of layer minimizes the internal covariate shift, which is the change in the input distribution to a learning system by applying a zero-mean and unit-variance transformation of the data while training the CNN model.

The output of the BN layer after every regular convolutional layer is followed by a nonlinear mapping called an activation function. This type of function is applied to each value in the feature maps of every convolutional layer. In our CNN, we use hyperbolic tangent (TanH) activation functions. Furthermore, to reduce the dimension of the activated large feature map volumes we use a max-pooling layer with a sliding window of size 3×3 and stride of 2. Fig. 2 depicts the size of filters in each convolutional layer as well as the dimension of their corresponding output feature maps.

3.3 Cross feature maps learning

To enhance the learning ability of our CNN, we use a cross feature maps learning block in our proposed architecture. From Fig. 2, we can notice that this block contains 128 1×1 convolutional filters in layer "Conv5" followed by a BN layer. These filters are used to learn a new association between the highest-level residual feature maps in the network. Additionally, this convolutional layer is the last layer before the classification block. Therefore, in order to keep the most representative features, we use an average-pooling layer that operates with a sliding window of size 3×3 and stride of 2. The output of this conceptual block is a feature maps volume of size 7×7×128 which takes the form of a fully-connected layer that is directly passed to a regular neural network.

3.4 Classification

To perform classification, we use a conceptual block that consists of three fully-connected layers. The first two layers contain 200 neurons followed by a TanH activation function. These two layers are used to learn deeper classification features in CNN. Finally, the number of neurons in the last fully-connected layer, also called classification layer, corresponds to the number of classes defined in Section 2.1. The classification layer is followed by a softmax activation function which maps the deepest features of the network learned by this layer to probability values. Input images to our CNN will be assigned to the class associated with the highest activation value using an arg max operator.

4 EXPERIMENTS

4.1 General experimental setup

We evaluated the performance of our proposed generic approach to perform manipulation parameter estimation through a set of experiments. In total, we considered four different tampering operations: JPEG compression, resampling, median filtering and gaussian blurring. The goal of these experiments is to show that using our generic data-driven approach we can forensically estimate the parameter of different types of manipulations. This is done without requiring a forensic investigator to make substantial changes to our generic approach. To extract classification features directly from data, we used our proposed constrained CNN architecture depicted in Fig. 2.

To create our training and testing databases, we downloaded images from the publicly available Dresden Image Database [15]. We then created different experimental databases where each corresponds to one particular manipulation with different parameter values applied to images. Since in general, a CNN's performance is dependent of the size and quality of the training set [5, 31], we created a large dataset for every experiment. The smallest dataset used in any of these experiments consisted of 438, 112 grayscale images of size 256×256. To do this, for every experimental database, the training and testing data were collected from two separate sets of images where the green layer of the nine central 256×256 patches of every image was retained. These patches are then processed using the four underlying types of manipulations with different parameters.

When training each CNN, we set the batch size equal to 64 and the parameters of the stochastic gradient descent as follows: $momentum = 0.95$, $decay = 0.0005$, and a learning rate $\epsilon = 10^{-3}$ that decreases every 3 epochs, which is the number of times that every sample in the training data was trained, by a factor $\gamma = 0.5$. We trained the CNN in each experiment for 36 epochs. Note that training and testing are disjoint and CNNs were tested on separate testing datasets. Additionally, while training CNNs, their testing accuracies on a separate testing database were recorded every 1, 000 iterations to produce tables in this section. In all tables, unaltered images are denoted by the uppercase letter U.

We implemented all of our CNNs using the Caffe deep learning framework [18]. We ran our experiments using one Nvidia GeForce GTX 1080 GPU with 8GB RAM. The datasets used in this work were all converted to the lmdb format. In what follows, we present the results of all our experiments.

4.2 Resampling: Scaling factor estimation

Resampling editing operation is often involved in creating composite image forgeries, where the size or angle of one source image needs to be adjusted. In this set of experiments, we evaluated the ability of our CNN-based approach to estimating the scaling factor in resampled images. We rescaled these images using a bilinear interpolation. We consider two practical scenarios where an investigator can estimate either a scaling factor from a given known candidate parameter set or an arbitrary scaling factor in more realistic scenario.

4.2.1 Scaling factor estimation given known candidate set. In this experiment, we assume that the investigator knows that the forger used one of scaling factor values in a fixed set. Here, this set is $\Theta = \{50\%, 60\%, 70\%, \cdots, 150\%\}$. Note that 100% means no scaling applied to an image. Our estimate θ is the scaling factor denoted by s. In this simplified scenario, we cast the problem of estimating the scaling factor in resampled images as a classification problem. Thus, we assign each scaling factor to a unique class c_k. We used our CNN to distinguish between these different scaling factors. The output layer of CNN in Fig. 2 consists of 11 neurons.

Next, we created a training database that consisted of 1, 465, 200 grayscale patches of size 256×256 . To accomplish this, we randomly selected 14, 800 image from the Dresden database. These images were divided into 256×256 grayscale blocks as described in Section 4.1. We then used the above defined scaling factors s to generate the corresponding resampled images of each grayscale patch. Subsequently, we selected 505 images not used for the training to build our testing database that consisted of 49, 995 grayscale patches of size 256×256 in the same manner described above.

Table 1: Confusion matrix showing the parameter identification accuracy of our constrained CNN for resampling manipulation with different scaling factors s; True (rows) versus Predicted (columns).

Acc=98.40%	s=50%	s=60%	s=70%	s=80%	s=90%	s=100%	s=110%	s=120%	s=130%	s=140%	s=150%
s=50%	**95.89%**	3.34%	0.68%	0.07%	0.00%	0.02%	0.00%	0.00%	0.00%	0.00%	0.00%
s=60%	4.91%	**92.19%**	2.68%	0.13%	0.02%	0.00%	0.02%	0.02%	0.00%	0.00%	0.02%
s=70%	0.53%	2.40%	**96.68%**	0.26%	0.02%	0.00%	0.00%	0.00%	0.00%	0.09%	0.02%
s=80%	0.04%	0.15%	0.33%	**99.36%**	0.00%	0.00%	0.00%	0.02%	0.02%	0.07%	0.00%
s=90%	0.00%	0.00%	0.15%	0.02%	**99.71%**	0.00%	0.00%	0.04%	0.02%	0.04%	0.00%
s=100%	0.02%	0.00%	0.00%	0.02%	0.00%	**99.87%**	0.00%	0.00%	0.00%	0.09%	0.00%
s=110%	0.00%	0.00%	0.00%	0.04%	0.00%	0.00%	**99.74%**	0.02%	0.02%	0.15%	0.02%
s=120%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.02%	**99.56%**	0.04%	0.31%	0.07%
s=130%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	**99.71%**	0.24%	0.04%
s=140%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	**100%**	0.00%
s=150%	0.02%	0.00%	0.00%	0.00%	0.08%	0.00%	0.00%	0.00%	0.00%	0.33%	**99.65%**

Table 2: Confusion matrix showing the parameter identification accuracy of our constrained CNN for resampling manipulation with different scaling factor intervals; True (rows) versus Predicted (columns).

Acc=95.45%	$I_{45-55\%}$	$I_{55-65\%}$	$I_{65-75\%}$	$I_{75-85\%}$	$I_{85-95\%}$	$I_{95-105\%}$	$I_{105-115\%}$	$I_{115-125\%}$	$I_{125-135\%}$	$I_{135-145\%}$	$I_{145-155\%}$
$I_{45-55\%}$	**93.89%**	5.33%	0.38%	0.20%	0.02%	0.09%	0.02%	0.00%	0.02%	0.00%	0.04%
$I_{55-65\%}$	3.80%	**86.27%**	7.82%	1.98%	0.04%	0.09%	0.00%	0.00%	0.00%	0.00%	0.00%
$I_{65-75\%}$	0.29%	4.82%	**85.98%**	8.24%	0.40%	0.04%	0.00%	0.09%	0.09%	0.02%	0.02%
$I_{75-85\%}$	0.02%	0.24%	3.40%	**93.98%**	1.87%	0.04%	0.02%	0.16%	0.27%	0.00%	0.00%
$I_{85-95\%}$	0.00%	0.00%	0.04%	1.18%	**97.67%**	0.04%	0.87%	0.09%	0.04%	0.07%	0.00%
$I_{95-105\%}$	0.07%	0.00%	0.07%	0.18%	0.22%	**99.38%**	0.02%	0.00%	0.07%	0.00%	0.00%
$I_{105-115\%}$	0.00%	0.00%	0.00%	0.00%	0.09%	0.00%	**99.11%**	0.27%	0.47%	0.07%	0.00%
$I_{115-125\%}$	0.00%	0.00%	0.00%	0.07%	0.02%	0.07%	0.09%	**98.69%**	1.00%	0.07%	0.00%
$I_{125-135\%}$	0.00%	0.00%	0.00%	0.02%	0.00%	0.02%	0.00%	0.04%	**99.56%**	0.31%	0.04%
$I_{135-145\%}$	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	1.47%	**97.24%**	1.29%
$I_{145-155\%}$	0.00%	0.00%	0.00%	0.00%	0.02%	0.00%	0.00%	0.00%	0.33%	1.44%	**98.20%**

We then used our trained CNN to estimate the scaling factor associated with each image in our testing database. We present our experimental results in Table 1 where the diagonal entires of the confusion matrix correspond to the estimation accuracy of each scaling factor using our CNN-based approach. Experiments show that our proposed approach can achieve 98.40% estimation accuracy which is equivalent to the identification rate of our CNN. Typically it can achieve higher than 99% on most scaling factors. Noticeably, our approach can detect 140% upscaled images with 100% accuracy. However, from Table 1 one can observe that the estimation accuracy decreases with low scaling factors. Specifically, when $s \leq 70\%$ our approach can achieve 96.68% with 70% downscaled images and at least 92.19% with 60% downscaled images.

Extracting resampling traces in downscaled images is very challenging problem since most of pixel value relationships are destroyed after an image is being downscaled. Specifically, our approach can estimate the scaling factor in 50% downscaled images with 95.89% accuracy. This result demonstrates that CNN can still extract good low-level pixel-value dependency features even in very challenging scenarios.

4.2.2 Estimation given arbitrary scaling factor. In our previous experiment, we showed that our CNN can distinguish between traces induced by different scaling factors in resampled images. In more realistic scenarios, the forger could use an arbitrary scaling factor. In this experiment, we assume that the investigator knows only an upper and lower bound on the scaling factor, i.e., $\Theta = [45\%, 155\%]$ is the parameter set and $\Phi = \{[45\%, 55\%), \cdots, [145\%, 155\%]\}$ is the set of all parameter subsets ϕ_k. Our estimate θ is the scaling factor denoted by s. Additionally, we assume that any $\theta \in \phi_k$ will be mapped to the centroid of ϕ_k using the operator $h(\cdot)$ defined in Section 2.1, i.e., if $s \in [t_k, t_{k+1})$ then $\hat{\theta} = \frac{t_{k+1}+t_k}{2}$. Each scaling interval will correspond to a class c_k. We use our CNN to distinguish between these scaling factor intervals. The output layer of CNN in Fig. 2 consists of 11 neurons.

We then built a training data that consisted of 732,600 grayscale 256×256 patches. To do this, we randomly selected 7,400 images from the Dresden database. Subsequently, we divided these images into 256×256 patches to generate grayscale images in the same manner described above. In order to generate the corresponding resampled images for each grayscale patch, we used the 'randint' command from the 'numpy' module in Python, which returns integers from the discrete uniform distribution, to compute scaling factor values that lie in the [45%, 155%] interval. We then selected

Table 3: Confusion matrix showing the parameter identification accuracy of our constrained CNN for JPEG compression manipulation with different quality factors (QF); True (rows) versus Predicted (columns).

Acc=98.90%	U	QF=50	QF=60	QF=70	QF=80	QF=90
U	**98.50%**	0.08%	0.19%	0.11%	0.36%	0.75%
QF=50	0.01%	**99.86%**	0.13%	0.00%	0.00%	0.00%
QF=60	0.01%	0.29%	**99.58%**	0.07%	0.05%	0.00%
QF=70	0.04%	0.23%	0.12%	**99.17%**	0.34%	0.11%
QF=80	0.05%	0.12%	0.54%	0.19%	**98.87%**	0.23%
QF=90	0.57%	0.12%	0.41%	0.60%	0.89%	**97.41%**

505 images not used for the training to similarly build our testing database which consisted of 49, 995 grayscale 256×256 patches.

We used our trained CNN to estimate the scaling factor interval of each testing patch in our testing dataset. In Table 2, we present the confusion matrix of our CNN used to estimate the different scaling factor intervals. Our experimental results show that our proposed approach can achieve 95.45% estimation accuracy. Typically it can achieve higher than 93% accuracy on most scaling factor intervals. From Table 2, one can notice that CNN can detect upscaled images using $s \in [125\%, 135\%)$ with 99.56% accuracy. Similarly to the previous experiment, the performance of CNN decreases with downscaled images when the scaling factor lies in intervals with small boundaries. Specifically, when $s < 95\%$ our approach can achieve 97.67% estimation accuracy with $s \in [85\%, 95\%)$ and at least 85.98% accuracy with $s \in [65\%, 75\%)$.

Similarly to the previous experiment, these results demonstrate again that even in challenging scenarios where images are downscaled with very small parameter values CNN can still extract good classification features to distinguish between the different used intervals. Noticeably, one can observe from Table 2 that CNN can determine resampled images using $s \in [45\%, 55\%)$ with 93.89% accuracy. Note that given that the chosen intervals are separate by just 1%, estimating an arbitrary scaling factor that lies in different intervals is more challenging than when the scaling factor estimate belongs to a fixed set of known candidates.

4.3 JPEG Compression: Quality factor estimation

JPEG is one of the most widely used image compression formats today. In this part of our experiments, we would like to estimate the quality factor of JPEG compressed images. To do this, we consider two practical scenarios where an investigator can estimate either a quality factor from a given known candidate parameter set or an arbitrary quality factor in more realistic scenario.

4.3.1 Quality factor estimation given known candidate set. In this experiment, we assume that the investigator knows that the forger used one of quality factor values in a fixed set. Here, this set is $\Theta = \{50, 60, 70, 80, 90\}$. Our estimate θ is the quality factor denoted by QF. In this simplified scenario, we approximate the quality factor estimation problem in JPEG compressed images by a classification problem. Thus, we assign each quality factor to a unique class c_k and the unaltered images class is denoted by c_0. The number of classes c_k's is equal to six which corresponds to the number of neurons in the output layer of CNN.

We built a training database that consisted of 777, 600 grayscale patches of size 256×256. First, we randomly selected 14, 400 images from the Dresden database. Next, we divided these images into 256×256 grayscale patches in the same manner described in Section 4.1. Each patch corresponds to a new image that has its corresponding tampered images created by the five different choices of JPEG quality factor.

To evaluate the performance of our proposed approach, we similarly created a testing database that consisted of 50, 112 grayscale patches. This is done by dividing 928 images not used for the training into 256×256 grayscale patches in the same manner described above. Then we applied to these grayscale patches the same editing operations.

We used our trained CNN to estimate the quality factor of each JPEG compressed patch in our testing dataset. In Table 3, we present the confusion matrix of our CNN-based approach used to estimate the different quality factors. The overall estimation accuracy on the testing database is 98.90%. One can observe that CNN can estimate the quality factor of JPEG compressed images with an accuracy typically higher than 98%. This demonstrates the ability of the constrained convolutional layer to adaptively extract low-level pixel-value dependency features directly from data. This also demonstrates that every quality factor induces detectable unique traces.

From Table 3, we can notice that the estimation accuracy of CNN decreases when the quality factor is high. More specifically, with $QF = 90$ images are 0.89% misclassified as JPEG compressed images with $QF = 80$ and 0.57% are misclassified as unaltered images. Similarly, with $QF = 80$ subject images are 0.54% misclassified as JPEG compressed images with $QF = 60$ and the unaltered images are 0.75% misclassified as JPEG compressed images with $QF = 90$.

4.3.2 Estimation given arbitrary quality factor. In the previous experiment, we experimentally demonstrated that CNN can distinguish between traces left by different JPEG quality factors. Similarly to the resampling experiments, we would like to estimate the JPEG quality factor in more realistic scenarios where the forger could use an arbitrary quality factor. we assume that the investigator knows only an upper and lower bound on the quality factor, i.e., $\Theta = [45, 100\%]$ is the parameter set and $\Phi = \{[45, 55), \cdots, [85, 95), [95, 100]\}$ is the set of all parameter subsets ϕ_k. Our estimate θ is the quality factor denoted by QF. Additionally, we assume that any $\theta \in \phi_k$ will be mapped to the centroid of ϕ_k using the operator $h(\cdot)$ defined in Section 2.1, i.e., if $QF \in [t_k, t_{k+1})$ then $\hat{\theta} = \frac{t_{k+1}+t_k}{2}$. We define the centroid of the inclusive interval $[95, 100]$ as 97. Each quality

Table 4: Confusion matrix showing the parameter identification accuracy of our constrained CNN for JPEG compression manipulation with different quality factors (QF) intervals; True (rows) versus Predicted (columns).

Acc=95.27%	**QF=45-54**	**QF=55-64**	**QF=65-74**	**QF=75-84**	**QF=85-94**	**QF=95-100**
QF=45-54	**96.76%**	3.23%	0.00%	0.00%	0.00%	0.01%
QF=55-64	2.39%	**95.20%**	2.32%	0.01%	0.00%	0.07%
QF=65-74	0.22%	2.20%	**94.49%**	3.03%	0.01%	0.05%
QF=75-84	0.19%	0.45%	2.83%	**94.46%**	1.93%	0.14%
QF=85-94	0.11%	0.49%	1.23%	2.54%	**94.01%**	1.62%
QF=95-100	0.07%	0.38%	0.53%	0.63%	1.68%	**96.71%**

Table 5: Confusion matrix showing the parameter identification accuracy of our constrained CNN for median filtering manipulation with different kernel sizes K_{size}; True (rows) versus Predicted (columns).

Acc=99.55%	**U**	$\mathbf{K_{size}}$ **= 3×3**	$\mathbf{K_{size}}$ **= 5×5**	$\mathbf{K_{size}}$ **= 7×7**	$\mathbf{K_{size}}$ **= 9×9**	$\mathbf{K_{size}}$ **= 11×11**	$\mathbf{K_{size}}$ **= 13×13**	$\mathbf{K_{size}}$ **= 15×15**
U	**99.97%**	0.00%	0.00%	0.02%	0.02%	0.00%	0.00%	0.00%
$\mathbf{K_{size}}$ **= 3×3**	0.02%	**99.92%**	0.03%	0.02%	0.02%	0.00%	0.00%	0.00%
$\mathbf{K_{size}}$ **= 5×5**	0.02%	0.02%	**99.86%**	0.10%	0.02%	0.00%	0.00%	0.00%
$\mathbf{K_{size}}$ **= 7×7**	0.02%	0.00%	0.08%	**99.60%**	0.27%	0.03%	0.00%	0.00%
$\mathbf{K_{size}}$ **= 9×9**	0.00%	0.00%	0.00%	0.03%	**99.68%**	0.29%	0.00%	0.00%
$\mathbf{K_{size}}$ **= 11×11**	0.00%	0.00%	0.00%	0.00%	0.26%	**99.36%**	0.38%	0.00%
$\mathbf{K_{size}}$ **= 13×13**	0.00%	0.00%	0.00%	0.00%	0.03%	0.50%	**98.82%**	0.66%
$\mathbf{K_{size}}$ **= 15×15**	0.00%	0.00%	0.00%	0.00%	0.02%	0.02%	0.70%	**99.26%**

factor interval will correspond to a class c_k. We use our CNN to distinguish between these quality factor intervals. The output layer of CNN in Fig. 2 consists of six neurons.

To train our CNN, we built a training database that consisted of 388, 800 grayscale patches of size 256×256. To do this, we randomly selected 7, 200 images from the Dresden database that we divided into 256×256 grayscale patches as described in Section 4.1. To generate for each grayscale patch its corresponding compressed images with quality factors that lie in the defined [45, 100] interval, similarly to the resampling experiments we used the 'randint' command from the 'numpy' module in Python to compute such quality factor values. Subsequently, we selected 928 images not used for the training to similarly build our testing database that consisted of 50, 112 grayscale patches of size 256×256.

We used our trained CNN to estimate the quality factor interval of each JPEG compressed patch in our testing dataset. In Table 4, we present the confusion matrix of our CNN used to estimate the different quality factor intervals. The overall estimation accuracy on the testing database is 95.92%. One can observe that CNN can estimate the quality factor interval of JPEG compressed images with an accuracy typically higher than 94%. This demonstrates again the ability of the constrained convolutional layer to adaptively extract low-level pixel-value dependency features directly from data to distinguish between quality factor intervals.

From Table 4, we can notice that the estimation accuracy is high either with low or high interval boundaries. Specifically, our approach can noticeably achieve 96.76% estimation accuracy when subject images are compressed with $QF \in [45, 54]$ and 96.71% accuracy with $QF \in [95, 100]$. This is mainly because these two intervals are either only followed by an upper interval or only preceded by a lower interval. Estimating an arbitrary quality factor that lies in different intervals is more challenging than when the quality factor estimate belongs to a fixed set of known candidates given that these intervals are chosen to be separate by $QF = 1$. Thus, when $55 \leq QF < 95$ intervals are misclassified as either a subsequent or preceding intervals (see Table 4).

4.4 Median Filtering: Kernel size estimation

Median filtering is a commonly used image smoothing technique, which is particularly effective for removing impulsive noise. It can also be used to hide artifacts of JPEG compression [35] and resampling [23]. This type of filter operates by using a sliding window, also called kernel, that keeps the median pixel value within the window dimension. When a forger applies a median filtering operation to an image, typically they choose an odd kernel size. Therefore, we assume that the investigator knows that the forger used one of kernel size values in a fixed set. Here, this set is $\Theta = \{3\times3, 5\times5, \cdots, 15\times15\}$. Our estimate θ is the kernel size denoted by k_{size}. In this simplified scenario, we approximate the filtering kernel size estimation problem in filtered images by a classification problem. Thus, we assign each choice of kernel size to a unique class c_k and the unaltered images class is denoted by c_0. The number of classes c_k's is equal to eight which corresponds to the number of neurons in the output layer of CNN.

We collected 15, 495 images for the training and testing datasets. We then randomly selected 14, 800 images from our experimental database for the training. Subsequently, we divided these images into 256×256 grayscale patches by retaining the green layer of the nine central blocks. As described above, each block will correspond to a new image that has its corresponding tampered images created

Table 6: Confusion matrix showing the parameter identification accuracy of our constrained CNN for gaussian blurring manipulation with different kernel sizes K_{size} and $\sigma = 0.3 \times ((K_{size} - 1) \times 0.5 - 1) + 0.8$; True (rows) versus Predicted (columns).

Acc=99.38%	**U**	**K_{size} = 3×3**	**K_{size} = 7×7**	**K_{size} = 11×11**	**K_{size} = 15×15**
U	**99.98%**	0.00%	0.00%	0.00%	0.02%
K_{size} = 3×3	0.00%	**99.96%**	0.02%	0.00%	0.02%
K_{size} = 7×7	0.00%	0.02%	**99.38%**	0.59%	0.02%
K_{size} = 11×11	0.00%	0.00%	0.07%	**98.61%**	1.32%
K_{size} = 15×15	0.00%	0.00%	0.00%	1.00%	**99.00%**

by median filtering manipulation using seven different kernel sizes. In total, our training database consisted of 1, 065, 600 patches.

We built our testing database in the same manner by dividing the 695 images not used for the training into 256×256 grayscale pixel patches. Then we edited these images to generate their tampered homologues. In total, our testing database consisted of 50, 040 patches. Our constrained CNN is then trained to determine unaltered images as well as the kernel size used to median filter testing images.

We used our trained CNN to estimate the median filtering kernel size of each filtered patch in our testing dataset. In Table 5, we present the confusion matrix of our CNN used to estimate the different kernel sizes. Our proposed approach can achieve 99.50% accuracy. From Table 5, we can notice that CNN can determine the kernel size with an estimation accuracy typically higher than 99%. Noticeably, it can achieve 99.97% with unaltered images and at least 98.82% with 13×13 median filtered images. Additionally, one can observe that most of the off-diagonal entries of the confusion matrix are equal to zero. Thus, we experimentally demonstrated that the constrained convolutional layer can adaptively extract traces left by a particular kernel size of a median filtering operation.

4.5 Gaussian Blurring

Gaussian filtering is often used for image smoothing, in order to remove noise or to reduce details. Similarly to median filtering, this type of filter operates by using a sliding window that convolves with all the regions of an image and has the following expression

$$G(x, y) = \alpha \exp\Big(-\frac{x^2 + y^2}{2\sigma^2}\Big), \tag{14}$$

where x and y are the kernel weight spatial locations where $G(0, 0)$ is the origin/central kernel weight, α is a scaling factor chosen such that $\sum_{x,y} G(x, y) = 1$, and σ^2 is the variance blur value. In this work, we experimentally investigate a set of two scenarios. First, we use CNN to estimate the filtering kernel size with size dependent blur variance. Subsequently, we fixed the kernel size and use our approach to identify the blur variance.

4.5.1 Kernel size estimation with size dependent blur variance. One of the most common ways to perform median blurring on images is to choose the kernel size with size dependent blur variance. That is, the blur variance is formally defined in programming libraries (e.g., OpenCV [17]) in terms of the kernel size as $\sigma^2 = \big(0.3 \times ((K_{size} - 1) \times 0.5 - 1) + 0.8\big)^2$. We assume that the investigator knows that the forger used one of kernel size values in a fixed set. Here, this set is $\Theta = \{3{\times}3, 7{\times}7, 11{\times}11, 15{\times}15\}$. Our estimate θ is the kernel size denoted by k_{size}. In this simplified scenario, we approximate the smoothing kernel size estimation problem in filtered images by a classification problem. Thus, we assign each choice of kernel size to a unique class c_k and the unaltered images class is denoted by c_0. The number of classes c_k's is equal to five which corresponds to the number of neurons in the output layer of CNN.

We collected 15, 920 images to perform the training and testing. To train CNN, we randomly selected 14, 800 images for training and the rest 1, 120 images were used for testing. Similarly to all previous experiments, images in the training and testing sets were divided into 256×256 blocks and the green layer of the central nine patches was retained. Each patch corresponds to a new image then we used the four different filtering kernel sizes to create their manipulated homologues. In total, we collect 666, 000 patches for training and 50, 400 for testing.

We used our trained CNN to estimate the smoothing kernel size of each filtered patch in our testing dataset. The confusion matrix of CNN to detect gaussian blur kernel size is presented in Table 6. Our proposed approach can identify the filtering kernel size with 99.38% accuracy. In particular it can identify unaltered images with 99.98% accuracy and it can achieve at least 98.64% detection rate with 11×11 filtered images. Furthermore, one can notice from Table 6 that the detection rate of the filtering kernel size decreases when the standard deviation blur $\sigma > 2$ which is equivalent of choosing a filtering kernel size bigger than 7×7. More specifically, the 11×11 filtered images are 1.32% misclassified as 15×15 filtered images. Similarly, the 15×15 filtered images are 1% misclassified as 11×11 filtered images. In what follows, we compare these results to the scenario when gaussian blurring is parameterized in terms of its variance σ^2 with a fixed filtering kernel size.

4.5.2 Variance Estimation with fixed kernel size. In this part, we use our approach to estimate the gaussian blur variance when the filtering kernel size is fixed to 5×5. To accomplish this, we assume that the investigator knows that the forger used one of blur standard deviation values in a fixed set. Here, this set is $\Theta = \{1, 2, 3, 4, 5\}$. Our estimate θ is the blur variance denoted by σ^2. In this simplified scenario, we approximate the blur variance estimation problem in smoothed images by a classification problem. Thus, we assign each choice of blur standard deviation σ a unique class c_k and the unaltered images class is denoted by c_0. The number of classes c_k's is equal to six which corresponds to the number of neurons in the output layer of CNN.

We collected 15, 734 images from our Dresden experimental database. To train the CNN, we then randomly selected 14, 800 images

Table 7: Confusion matrix showing the parameter identification accuracy of our constrained CNN for gaussian blurring manipulation with fixed kernel size (i.e., K_{size} = 5×5) and different σ; True (rows) versus Predicted (columns).

Acc=96.94%	**U**	$\sigma = 1$	$\sigma = 2$	$\sigma = 3$	$\sigma = 4$	$\sigma = 5$
U	**99.94%**	0.04%	0.00%	0.00%	0.00%	0.02%
$\sigma = 1$	0.01%	**99.90%**	0.08%	0.00%	0.00%	0.00%
$\sigma = 2$	0.00%	0.01%	**99.92%**	0.06%	0.01%	0.00%
$\sigma = 3$	0.00%	0.04%	0.12%	**97.87%**	1.70%	0.27%
$\sigma = 4$	0.01%	0.01%	0.02%	1.45%	**90.68%**	7.82%
$\sigma = 5$	0.02%	0.01%	0.01%	0.08%	6.54%	**93.33%**

for the training that we divided into 256×256 patches as described above. Then we generated their corresponding edited patches using the five possible parameter values. In total our training database consisted of 799, 200 patches. To evaluated our method in determining the gaussian blur variance σ^2, similarly we divided the 934 images not used for the training into 256×256 blocks then we generated their corresponding edited patches using the same editing operations. In total, we collected 50, 400 patches for the testing database.

We used our trained CNN to estimate the blur variance of each filtered patch in our testing dataset. In Table 7, we present the confusion matrix of our method. Our experimental results show that our proposed approach can determine the blur variance with 96.94%. From Table 7, we can notice from the confusion matrix of CNN that these results match the results presented in Table 6. In fact, when the standard deviation blur $\sigma \leq 2$, CNN can identify the parameter values with an accuracy higher than 99%. Noticeably, it can achieve 99.94% at identifying unaltered images and at least 99.90% accuracy with gaussian blurred images using a standard deviation blur $\sigma = 1$.

One can observe that similarly to the previous experiment, when $\sigma > 2$ the estimation accuracy significantly decreases and it can achieve at most 97.87% accuracy with gaussian blurred images using a standard deviation blur $\sigma = 3$. Note that in the size dependent blur variance experiment, the highest value of σ is equal to 2.6. Finally, these experiments demonstrate that CNN is able to adaptively extract good low-level representative features associated with every choice of the variance value.

4.6 Experimental results summary

In this section, we experimentally investigated the ability of our CNN-based generic approach to forensically estimate the manipulation parameters. Our experimental results showed that CNNs associated with the constrained convolutional layer are good candidates to extract low-level classification features and to estimate a particular manipulation parameter. We used the proposed CNN to capture pixel-value dependency traces induced by each different manipulation parameter in all our experiments. In a simplified scenario where a forensic investigator knows a priori a fixed set of parameter candidates, our CNN was able to perform manipulation parameter estimation with an accuracy typically higher than 98% with all underlying image editing operations.

Specifically, when the parameter value θ belongs to a fixed set of known candidates, CNN can accurately estimate resampling scaling factor, JPEG quality factor, median filtering kernel size and gaussian blurring kernel size respectively with 98.40%, 98.90%, 99.55% and 99.38% accuracy. This demonstrates also that our method is generic and could be used with multiple types of image manipulation. It is worth mentioning that when images are downscaled, scaling factor estimation is difficult [22]. Our proposed approach, however, is still able to determine the scaling factor in downscaled images with at least 92% accuracy.

When the parameter value θ is an arbitrary value in a bounded but countable set, our CNNs performance decreases. This is mainly because we consider a very challenging problem where parameter intervals are chosen to be separate by one unit distance, e.g. scaling factor interval [65%, 75%) followed by [75%, 85%) interval. Specifically, our generic approach can estimate the resampling scaling factor interval as well as the JPEG quality factor interval with an accuracy respectively equal to 95.45% and 95.27%. These results demonstrate the ability of CNN to distinguish between different parameter value intervals even when the distance between these intervals is very small.

Though we have demonstrated through our experiments that our proposed method can accurately perform manipulation parameter estimation, our goal is not necessarily to outperform existing parameter estimation techniques. It is instead to propose a new data-driven manipulation parameter estimation approach that can provide accurate manipulation parameter estimates for several different manipulations without requiring an investigator to analytically derive a new estimator for each manipulation.

5 CONCLUSION

In this paper, we have proposed a data-driven generic approach to performing forensic manipulation parameter estimation. Instead of relying on theoretical analysis of parametric models, our proposed method is able to learn estimators directly from a set of labeled data. Specifically, we cast the problem of manipulation parameter estimation as a classification problem. To accomplish this, we first partitioned the manipulation parameter space into an ordered set of disjoint subsets, then we assigned a class to each subset. Subsequently, we designed a CNN-based classifier which makes use of a constrained convolutional layer to learn traces left by a desired manipulation that has been applied using parameter values in each parameter subset. The ultimate goal of this work is to show that our generic parameter estimator can be used with multiple types of image manipulation without requiring a forensic investigator to make substantial changes to the proposed method. We evaluated the effectiveness of our generic estimator through a set of experiments using four different types of parameterized manipulation. The results of these experiments showed that our generic method can provide an estimate for these manipulations with estimation accuracies typically in the 95% to 99% range.

6 ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1553610. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

REFERENCES

[1] Bahrami, K., Kot, A. C., Li, L., and Li, H. Blurred image splicing localization by exposing blur type inconsistency. *IEEE Transactions on Information Forensics and Security 10*, 5 (May 2015), 999–1009.

[2] Bayar, B., and Stamm, M. C. On the robustness of constrained convolutional neural networks to jpeg post-compression for image resampling detection. In *The 2017 IEEE International Conference on Acoustics, Speech and Signal Processing*, IEEE.

[3] Bayar, B., and Stamm, M. C. A deep learning approach to universal image manipulation detection using a new convolutional layer. In *Proceedings of the 4th ACM Workshop on Information Hiding and Multimedia Security* (2016), ACM, pp. 5–10.

[4] Bayar, B., and Stamm, M. C. Design principles of convolutional neural networks for multimedia forensics. In *International Symposium on Electronic Imaging: Media Watermarking, Security, and Forensics* (2017), IS&T.

[5] Bengio, Y. Practical recommendations for gradient-based training of deep architectures. In *Neural Networks: Tricks of the Trade*. Springer, 2012, pp. 437–478.

[6] Bianchi, T., Rosa, A. D., and Piva, A. Improved dct coefficient analysis for forgery localization in jpeg images. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)* (May 2011), pp. 2444–2447.

[7] Chen, F., and Ma, J. An empirical identification method of gaussian blur parameter for image deblurring. *IEEE Transactions on signal processing 57*, 7 (2009), 2467–2478.

[8] Chen, J., Kang, X., Liu, Y., and Wang, Z. J. Median filtering forensics based on convolutional neural networks. *IEEE Signal Processing Letters 22*, 11 (Nov. 2015), 1849–1853.

[9] Cho, T. S., Paris, S., Horn, B. K. P., and Freeman, W. T. Blur kernel estimation using the radon transform. In *CVPR 2011* (June 2011), pp. 241–248.

[10] Conotter, V., Comesaa, P., and Prez-Gonzlez, F. Forensic detection of processing operator chains: Recovering the history of filtered jpeg images. *IEEE Transactions on Information Forensics and Security 10*, 11 (Nov 2015), 2257–2269.

[11] Cox, I. J., Kilian, J., Leighton, F. T., and Shamoon, T. Secure spread spectrum watermarking for multimedia. *IEEE Transactions on Image Processing 6*, 12 (Dec 1997), 1673–1687.

[12] Fan, Z., and De Queiroz, R. L. Identification of bitmap compression history: Jpeg detection and quantizer estimation. *IEEE Transactions on Image Processing 12*, 2 (2003), 230–235.

[13] Farid, H. Blind inverse gamma correction. *IEEE Transactions on Image Processing 10*, 10 (Oct 2001), 1428–1433.

[14] Fridrich, J., and Kodovskỳ, J. Rich models for steganalysis of digital images. *IEEE Transactions on Information Forensics and Security 7*, 3 (2012), 868–882.

[15] Gloe, T., and Böhme, R. The dresden image database for benchmarking digital image forensics. *Journal of Digital Forensic Practice 3*, 2-4 (2010), 150–159.

[16] Goljan, M., and Fridrich, J. Camera identification from cropped and scaled images. In *Electronic Imaging* (2008), International Society for Optics and Photonics, pp. 68190E–68190E.

[17] Itseez. Open source computer vision library. https://github.com/itseez/opencv, 2015.

[18] Jia, Y., Shelhamer, E., Donahue, J., Karayev, S., Long, J., Girshick, R., Guadarrama, S., and Darrell, T. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093* (2014).

[19] Kang, X., Stamm, M. C., Peng, A., and Liu, K. J. R. Robust median filtering forensics using an autoregressive model. *IEEE Transactions on Information Forensics and Security, 8*, 9 (Sept. 2013), 1456–1468.

[20] Kang, X., Stamm, M. C., Peng, A., and Liu, K. R. Robust median filtering forensics using an autoregressive model. *IEEE Transactions on Information Forensics and Security 8*, 9 (2013), 1456–1468.

[21] Kee, E., Johnson, M. K., and Farid, H. Digital image authentication from jpeg headers. *IEEE Transactions on Information Forensics and Security 6*, 3 (Sept. 2011), 1066–1075.

[22] Kirchner, M. Fast and reliable resampling detection by spectral analysis of fixed linear predictor residue. In *Proceedings of the 10th ACM Workshop on Multimedia and Security* (New York, NY, USA, 2008), MM&Sec '08, ACM, pp. 11–20.

[23] Kirchner, M., and Bohme, R. Hiding traces of resampling in digital images. *IEEE Transactions on Information Forensics and Security 3*, 4 (2008), 582–592.

[24] Kirchner, M., and Fridrich, J. On detection of median filtering in digital images. In *IS&T/SPIE Electronic Imaging* (2010), International Society for Optics and Photonics, pp. 754110–754110.

[25] Pevny, T., Bas, P., and Fridrich, J. Steganalysis by subtractive pixel adjacency matrix. *IEEE Transactions on Information Forensics and Security 5*, 2 (June 2010), 215–224.

[26] Pevny, T., and Fridrich, J. Detection of double-compression in jpeg images for applications in steganography. *IEEE Transactions on Information Forensics and Security 3*, 2 (June 2008), 247–258.

[27] Pfennig, S., and Kirchner, M. Spectral methods to determine the exact scaling factor of resampled digital images. In *Communications Control and Signal Processing (ISCCSP), 2012 5th International Symposium on* (2012), IEEE, pp. 1–6.

[28] Popescu, A. C., and Farid, H. Exposing digital forgeries by detecting traces of resampling. *IEEE Transactions on Signal Processing 53*, 2 (Feb. 2005), 758–767.

[29] Qiu, X., Li, H., Luo, W., and Huang, J. A universal image forensic strategy based on steganalytic model. In *Proceedings of the 2nd ACM workshop on Information hiding and multimedia security* (2014), ACM, pp. 165–170.

[30] Ruanaidh, J. J. O., and Pun, T. Rotation, scale and translation invariant spread spectrum digital image watermarking. *Signal processing 66*, 3 (1998), 303–317.

[31] Simard, P. Y., Steinkraus, D., and Platt, J. C. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR* (2003), vol. 3, pp. 958–962.

[32] Stamm, M. C., Chu, X., and Liu, K. J. R. Forensically determining the order of signal processing operations. In *IEEE International Workshop on Information Forensics and Security (WIFS)* (Nov 2013), pp. 162–167.

[33] Stamm, M. C., and Liu, K. J. R. Forensic detection of image manipulation using statistical intrinsic fingerprints. *IEEE Transactions on Information Forensics and Security 5*, 3 (Sept 2010), 492–506.

[34] Stamm, M. C., and Liu, K. J. R. Forensic estimation and reconstruction of a contrast enhancement mapping. In *2010 IEEE International Conference on Acoustics, Speech and Signal Processing* (March 2010), pp. 1698–1701.

[35] Stamm, M. C., and Liu, K. R. Anti-forensics of digital image compression. *IEEE Transactions on Information Forensics and Security 6*, 3 (2011), 1050–1065.

[36] Stamm, M. C., Wu, M., and Liu, K. J. R. Information forensics: An overview of the first decade. *IEEE Access 1* (2013), 167–200.

[37] Thai, T. H., Cogranne, R., Retraint, F., et al. Jpeg quantization step estimation and its applications to digital image forensics. *IEEE Transactions on Information Forensics and Security 12*, 1 (2017), 123–133.